P&S Modern SSDs

Fine-Grained Mapping & Multi-Plane Operation-Aware Block Management

Dr. Mohammad Sadrosadati
Prof. Onur Mutlu
ETH Zürich

Fall 2022

14 December 2022

Recap: What We Have Discussed So Far

- **SSD Organization**
- **NAND Flash Organization and Operations**
- **Advanced NAND Flash Commands**
- **FTL: Address Translation & Garbage Collection**

Today's Agenda

- Fine-Grained Mapping
- Multi-plane Operation-Aware Blk. Mgmt.

I/O Mismatch b/w OS and NAND Flash

- The page size (i.e., minimum I/O unit) of NAND flash memory has continuously increased
  - From 256 bytes to 16 KiB
  - Low area overhead and high bandwidth (size / latency)

- The logical block (or sector) size of file systems has also increased
  - From 512 bytes to 4 KiB
  - To more efficiently work with NAND flash-based SSDs
  - Increasing the block size is not straightforward
    - I/O handling is closely related to OS memory management
    - Memory page size = 4 KiB
    - Unnecessary fetch or eviction at the page cache

Small Write Requests

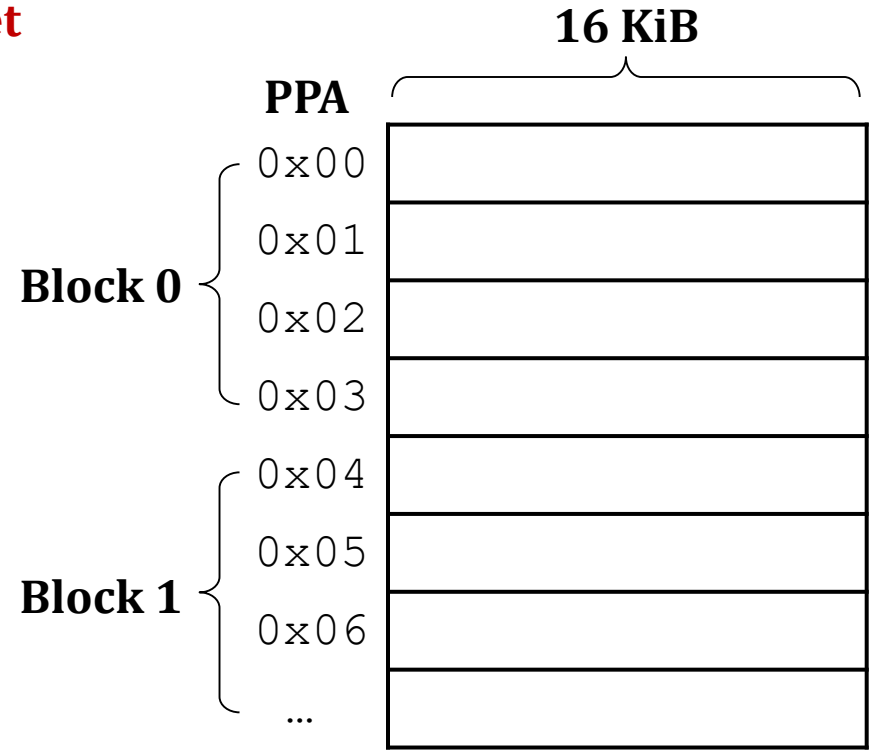
- Inefficiencies due to the erase-before-write property

Req (LBA: 0x04, Size: 1, DIR: W, Data: A)

0b 0000 0000 0000 0100

16-KiB Page Number **4-KiB Offset**

LPA	PPA
0x00	–
0x01	–
0x02	–
0x03	–
0x04	–
0x05	–
…	…



Small Write Requests

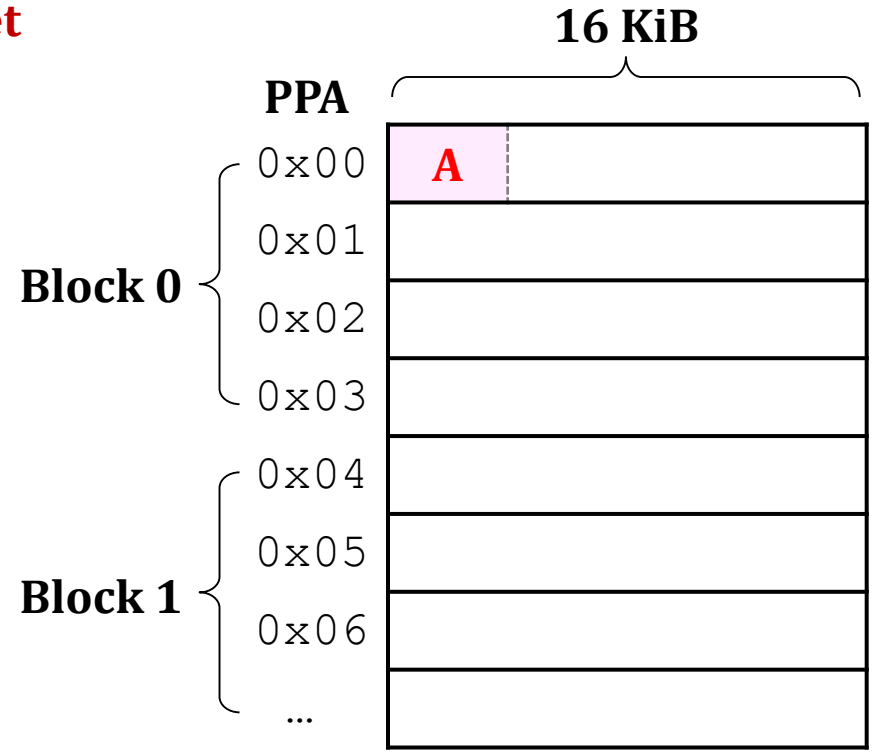
- Inefficiencies due to the erase-before-write property

Req (LBA: 0x04, Size: 1, DIR: W, Data: A)

0b 0000 0000 0000 0100

16-KiB Page Number **4-KiB Offset**

LPA	PPA
0x00	–
0x01	–
0x02	–
0x03	–
0x04	–
0x05	–
…	…



Small Write Requests

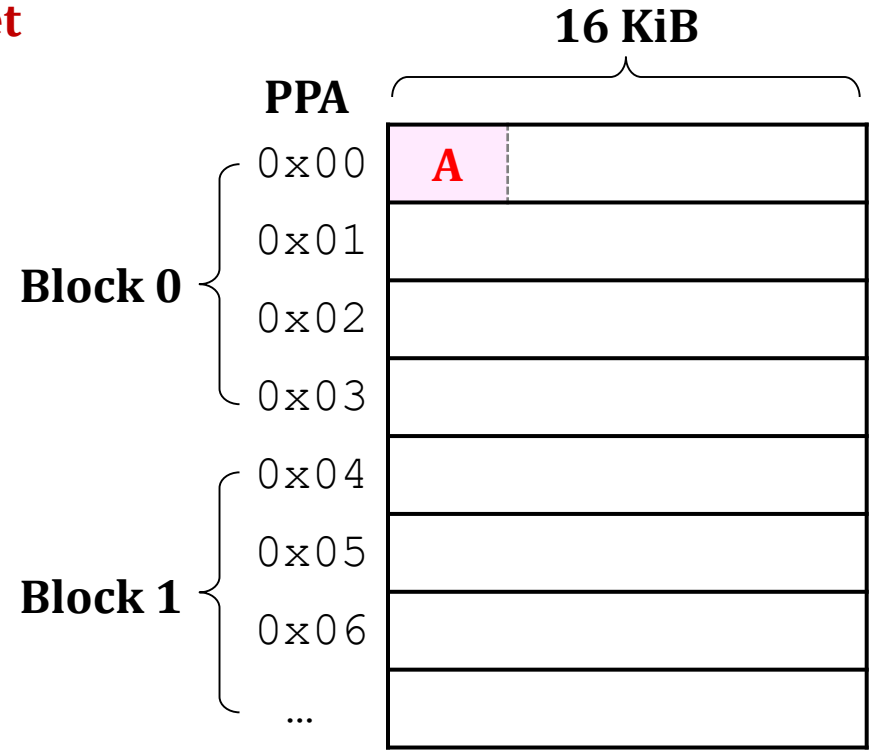
- Inefficiencies due to the erase-before-write property

Req (LBA: 0x04, Size: 1, DIR: W, Data: A)

0b 0000 0000 0000 0100

16-KiB Page Number **4-KiB Offset**

LPA	PPA
0x00	–
0x01	0x00
0x02	–
0x03	–
0x04	–
0x05	–
…	…



Small Write Requests

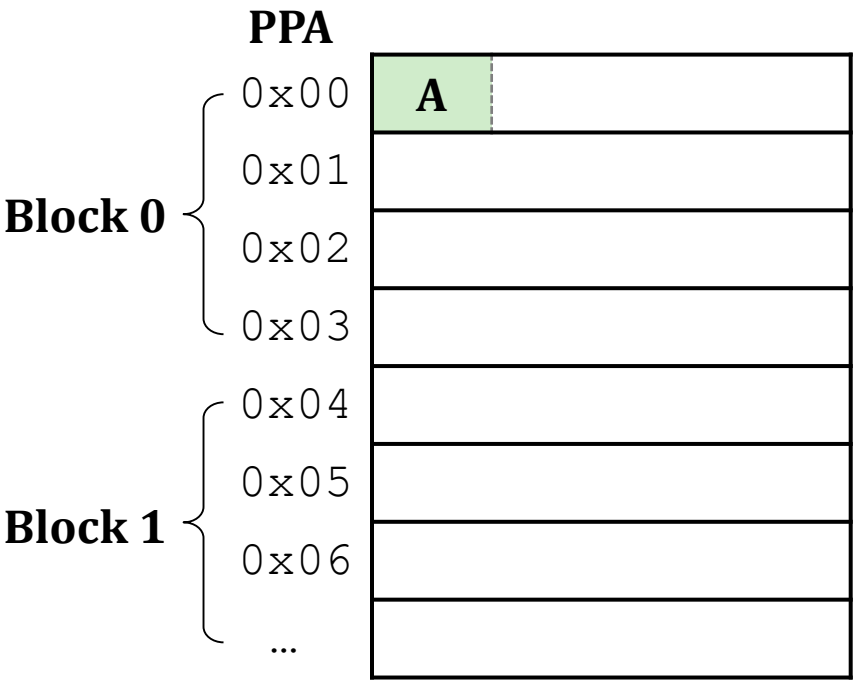
- Inefficiencies due to the erase-before-write property

Req (**LBA:** `0x01`, **Size:** `2`, **DIR:** `W`, **Data:** B, C)

`0b 0000 0000 0000 0001`

16-KiB Page Number **4-KiB Offset**

LPA	PPA
0x00	–
0x01	0x00
0x02	–
0x03	–
0x04	–
0x05	–
…	…



Small Write Requests

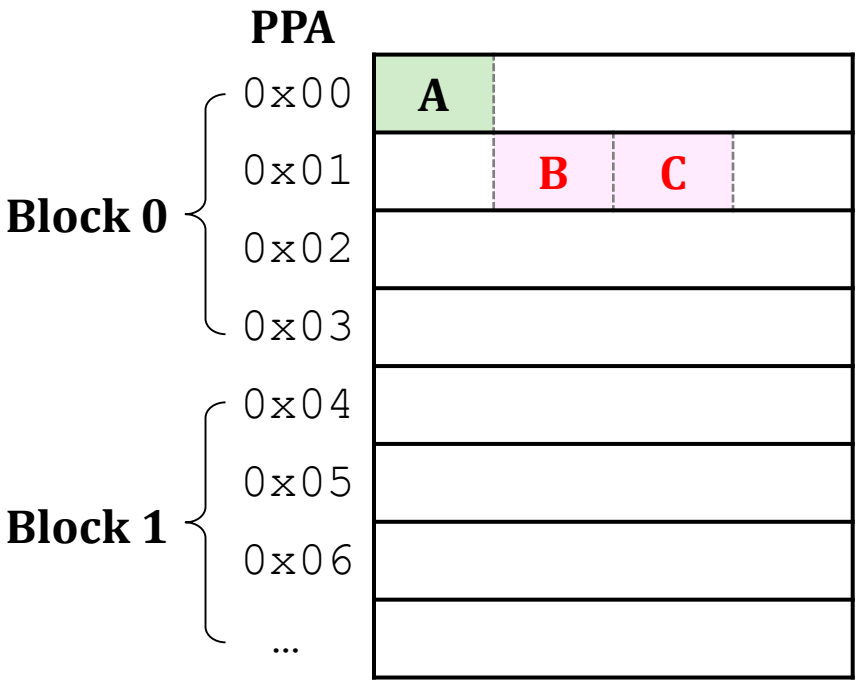
- Inefficiencies due to the erase-before-write property

Req (**LBA:** `0x01`, **Size:** `2`, **DIR:** `W`, **Data:** B, C)

`0b 0000 0000 0000 0001`

16-KiB Page Number **4-KiB Offset**

LPA	PPA
0x00	–
0x01	0x00
0x02	–
0x03	–
0x04	–
0x05	–
…	…



Small Write Requests

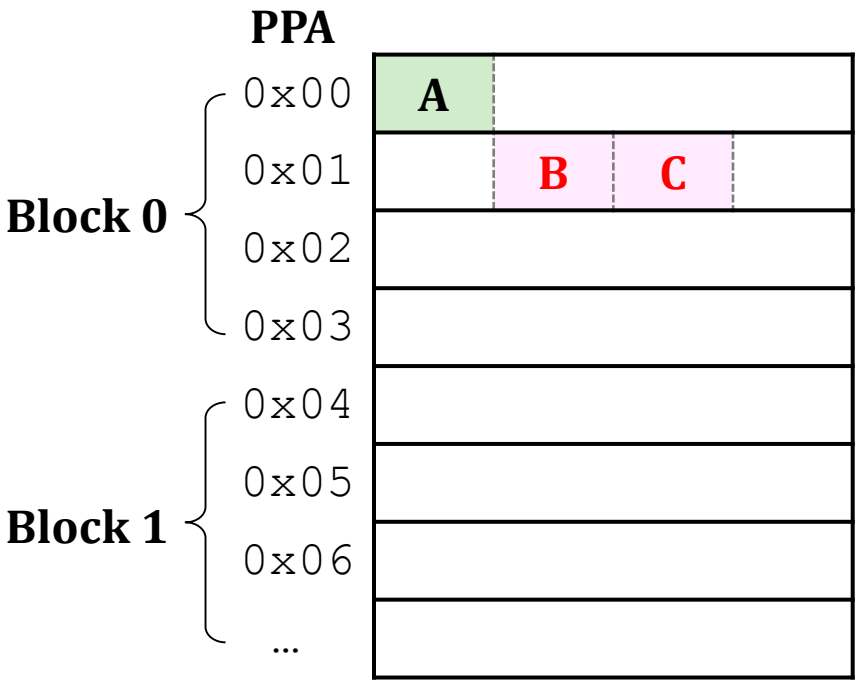
- Inefficiencies due to the erase-before-write property

Req (**LBA:** `0x01`, **Size:** `2`, **DIR:** `W`, **Data:** B, C)

`0b 0000 0000 0000 0001`

16-KiB Page Number **4-KiB Offset**

LPA	PPA
0x00	0x01
0x01	0x00
0x02	–
0x03	–
0x04	–
0x05	–
…	…



Small Write Requests

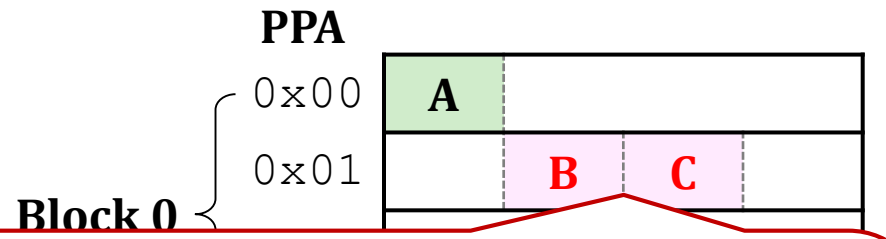
- Inefficiencies due to the erase-before-write property

Req (**LBA:** `0x01`, **Size:** `2`, **DIR:** `W`, **Data:** **B, C**)

`0b 0000 0000 0000 0001`

16-KiB Page Number **4-KiB Offset**

LPA	PPA
0x00	**0x01**
0x01	0x00
0x02	–
0x03	–
0x04	–
0x05	–
…	…



1. **Why at the middle of the page?**
   - To keep the 4-KiB offset: mapping table stores only the index of the 16-KiB page!
2. **Why not using the unused space in physical page `0x00`?**
   - That space is already mapped to logical pages 0x05~0x07 (not written yet).

Small Write Requests

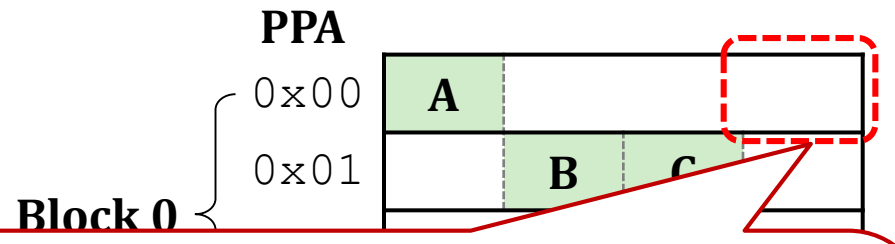
- Inefficiencies due to the erase-before-write property

Req (LBA: 0x07, Size: 1, DIR: W, Data: D)

`0b 0000 0000 0000 0111`

16-KiB Page Number **4-KiB Offset**

LPA	PPA
0x00	0x01
0x01	0x00
0x02	–
0x03	–
0x04	–
0x05	–
…	…



Q: Can we use the unused space?

A: Not likely, because

- Data randomization – Cells in the unused space have been already programmed.
- Program-order constraint – Re-programming physical page `0x00` can affect the reliability of the data stored in physical page `0x01`.

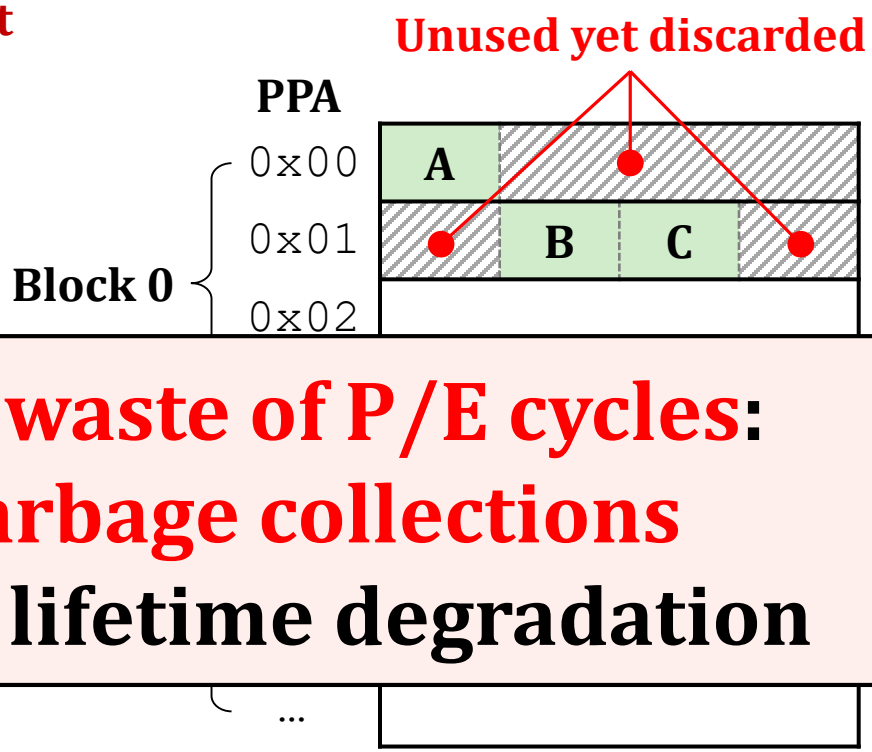
Small Write Requests

- Inefficiencies due to the erase-before-write property

Req (LBA: 0x07, Size: 1, DIR: W, Data: D)

```
0b 0000 0000 0000 0111
   16-KiB Page Number  4-KiB Offset
```

LPA	PPA
0x00	0x01
0x01	0x00
...	...



Small writes cause waste of P/E cycles:
More frequent garbage collections
→ Performance and lifetime degradation

Small Write Requests

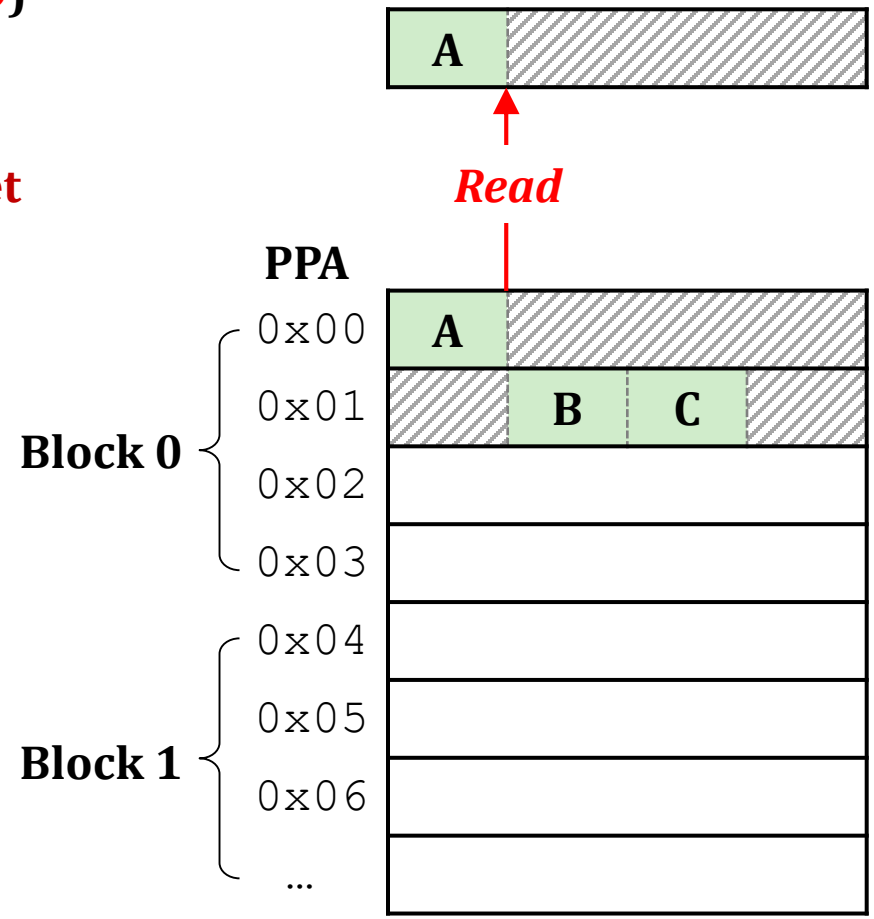
- Inefficiencies due to the erase-before-write property

Req (LBA: 0x07, Size: 1, DIR: W, Data: D)

0b 0000 0000 0000 0111

16-KiB Page Number **4-KiB Offset**

LPA	PPA
0x00	0x01
0x01	0x00
0x02	–
0x03	–
0x04	–
0x05	–
…	…



Small Write Requests

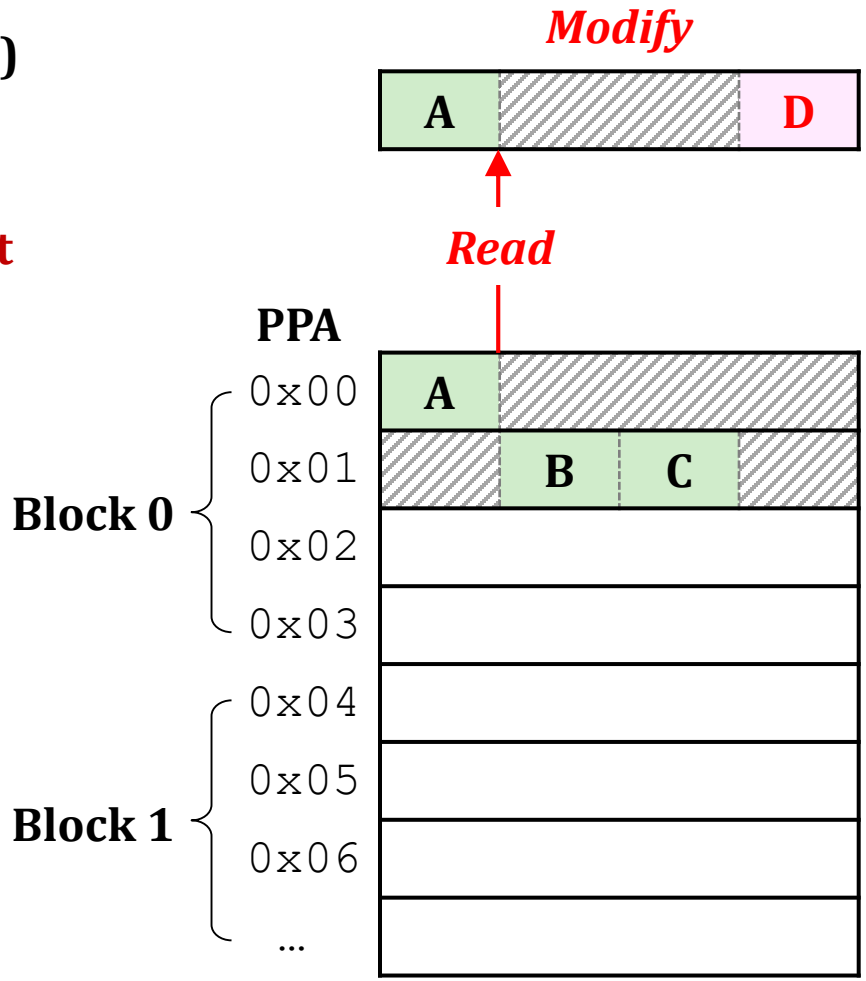
- Inefficiencies due to the erase-before-write property

Req (LBA: 0x07, Size: 1, DIR: W, Data: D)

`0b 0000 0000 0000 0111`

16-KiB Page Number **4-KiB Offset**

LPA	PPA
0x00	0x01
0x01	0x00
0x02	–
0x03	–
0x04	–
0x05	–
…	…



Small Write Requests

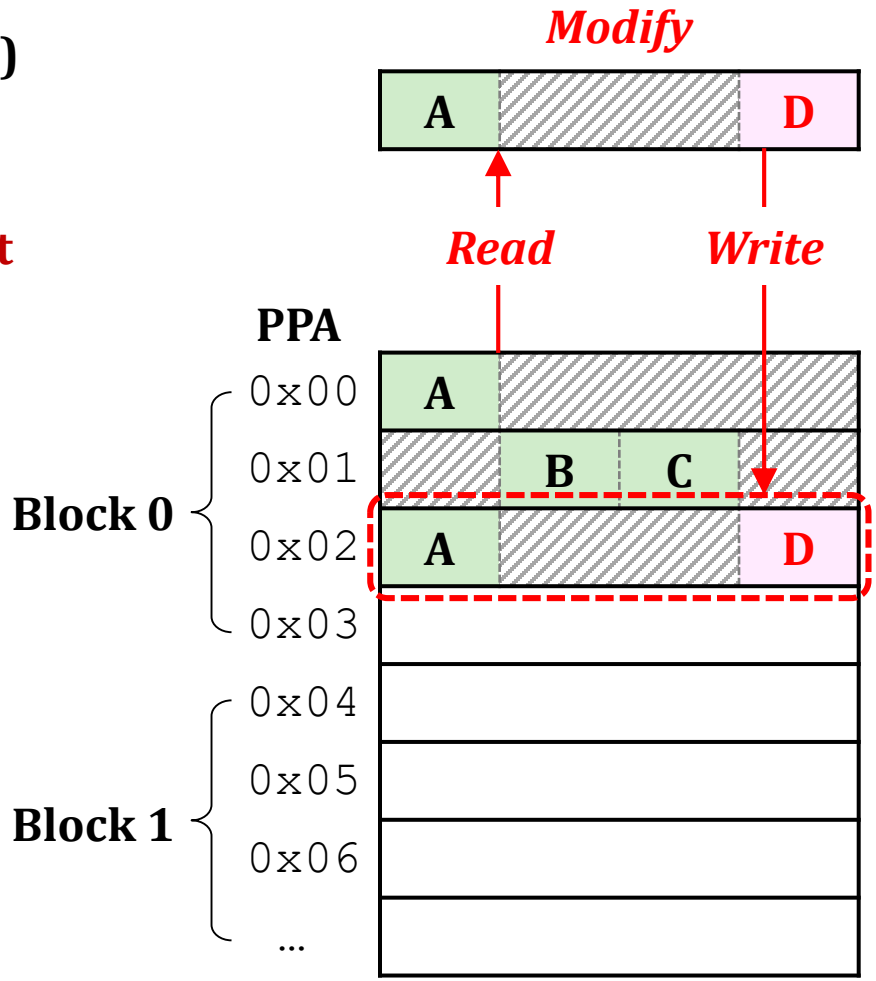
- Inefficiencies due to the erase-before-write property

Req (**LBA:** `0x07`, **Size:** `1`, **DIR:** `W`, **Data:** D)

`0b 0000 0000 0000 0111`

16-KiB Page Number **4-KiB Offset**

LPA	PPA
0x00	0x01
0x01	0x00
0x02	–
0x03	–
0x04	–
0x05	–
…	…



Small Write Requests

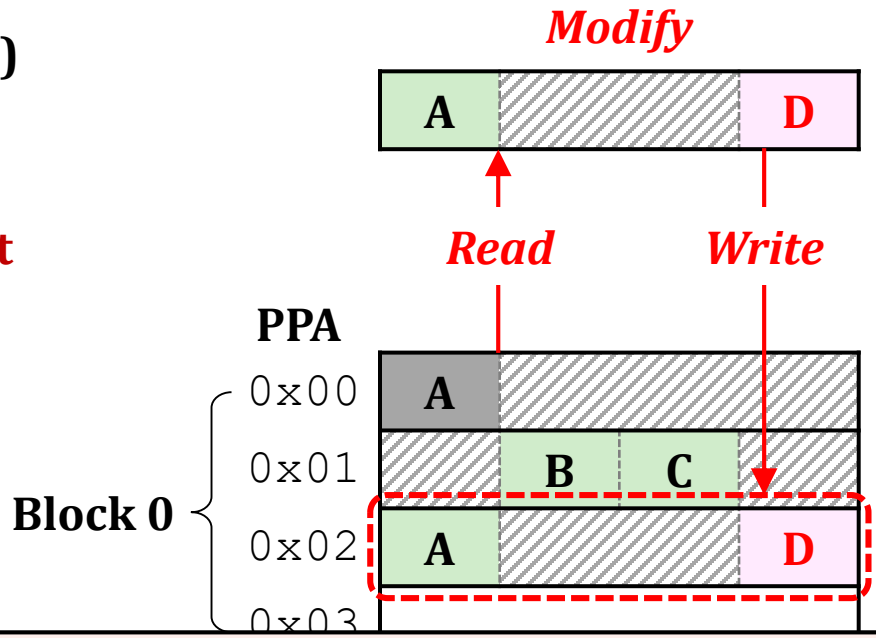
- Inefficiencies due to the erase-before-write property

Req (**LBA:** `0x07`, **Size:** `1`, **DIR:** `W`, **Data:** D)

`0b 0000 0000 0000 0111`

16-KiB Page Number | 4-KiB Offset

LPA	PPA
0x00	0x01
0x01	0x02
0x02	–



Small writes cause read-modify-writes:
Waste of P/E cycles + additional read operations
→ Performance and lifetime degradation

Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

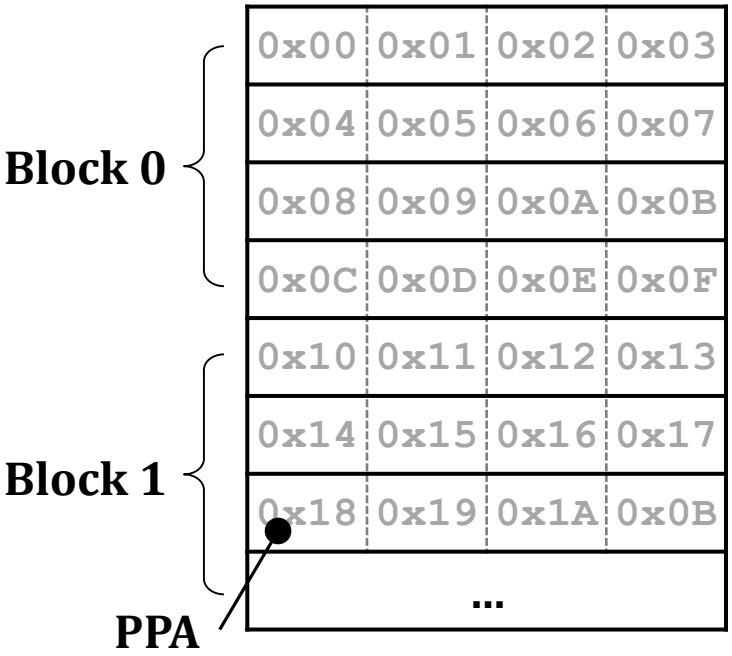
Req (LBA: 0x04, Size: 1, DIR: W, Data: A)

Req (LBA: 0x01, Size: 2, DIR: W, Data: B, C)

Req (LBA: 0x07, Size: 1, DIR: W, Data: D)

Page Buffer

LPA	PPA
0x00	–
0x01	–
…	–
0x04	–
…	–
0x07	–
…	…



Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

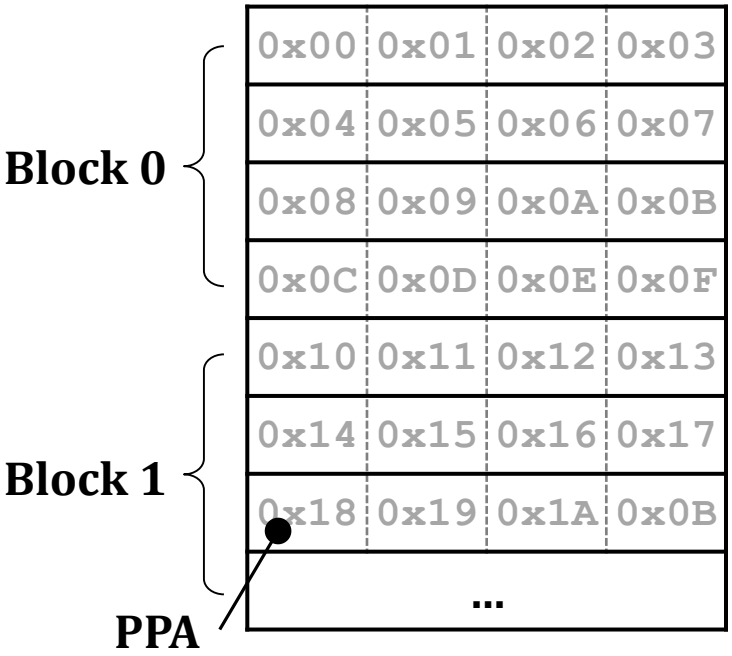
Req (LBA: 0x04, Size: 1, DIR: W, Data: A)

Req (LBA: 0x01, Size: 2, DIR: W, Data: B, C)

Req (LBA: 0x07, Size: 1, DIR: W, Data: D)

Page Buffer

LPA	PPA
0x00	–
0x01	–
…	–
0x04	–
…	–
0x07	–
…	…



Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

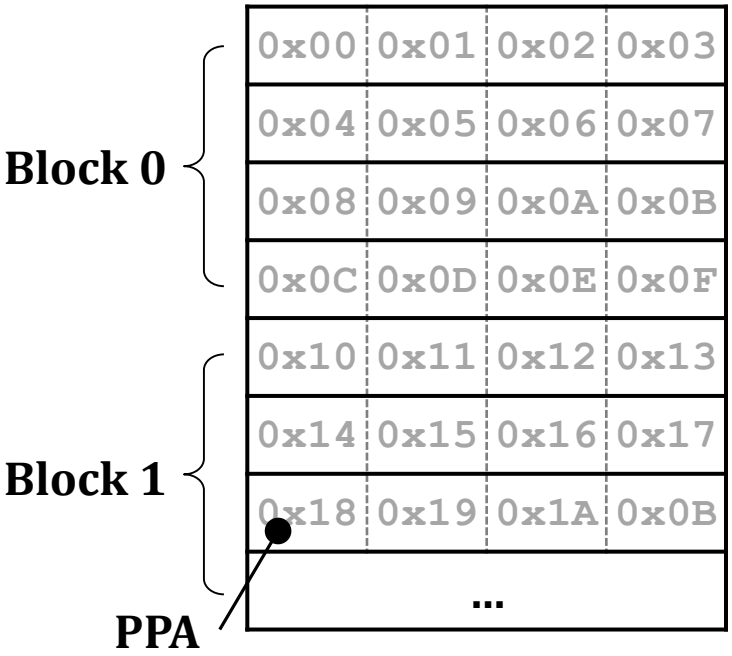
Req (LBA: 0x04, Size: 1, DIR: W, Data: A)

Req (LBA: 0x01, Size: 2, DIR: W, Data: B, C)

Req (LBA: 0x07, Size: 1, DIR: W, Data: D)

Page Buffer

LPA	PPA
0x00	–
0x01	–
…	–
0x04	0x00
…	–
0x07	–
…	…



Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

Req (LBA: 0x04, Size: 1, DIR: W, Data: A)

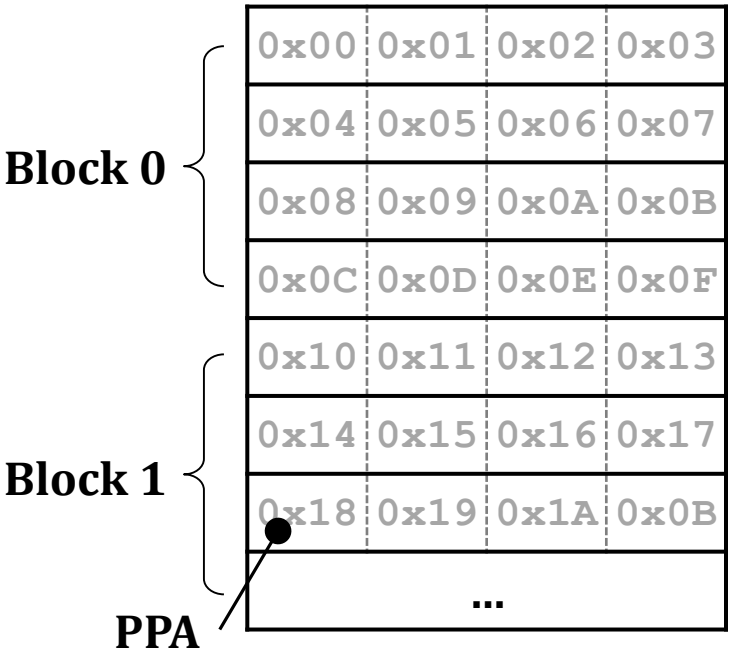
Req (LBA: 0x01, Size: 2, DIR: W, Data: B, C)

Req (LBA: 0x07, Size: 1, DIR: W, Data: D)

Page Buffer

A			

LPA	PPA
0x00	–
0x01	–
…	–
0x04	0x00
…	–
0x07	–
…	…



Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

Req (LBA: 0x04, Size: 1, DIR: W, Data: A)

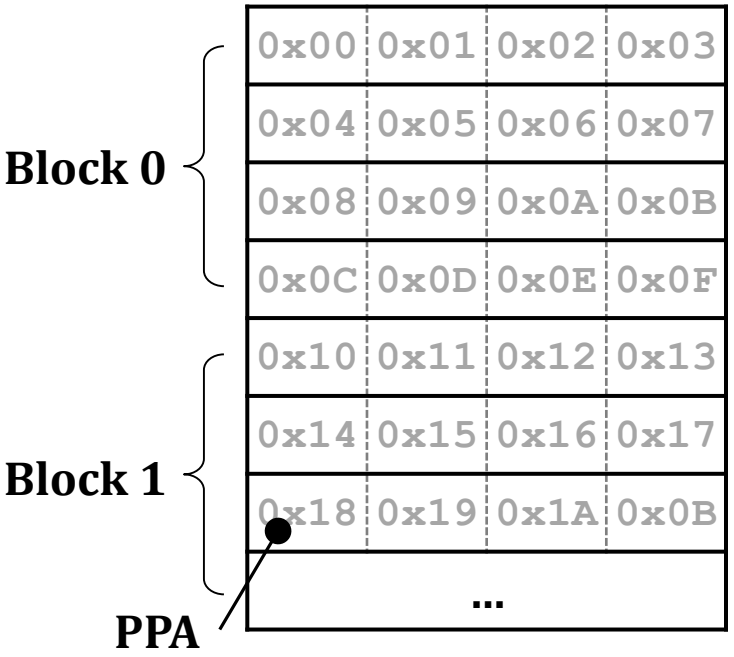
Req (LBA: 0x01, Size: 2, DIR: W, Data: B, C)

Req (LBA: 0x07, Size: 1, DIR: W, Data: D)

Page Buffer

A	B	C	

LPA	PPA
0x00	–
0x01	–
…	–
0x04	0x00
…	–
0x07	–
…	…



Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

Req (LBA: 0x04, Size: 1, DIR: W, Data: A)

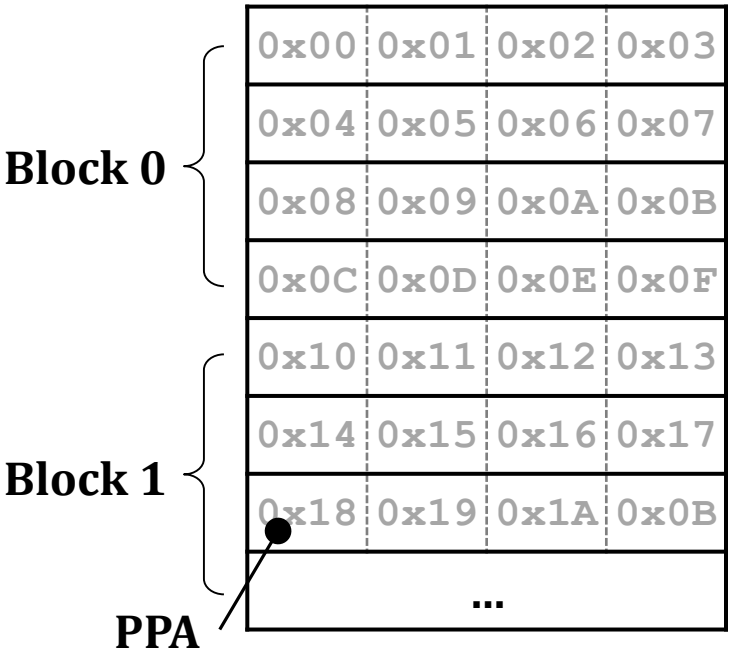
Req (LBA: 0x01, Size: 2, DIR: W, Data: B, C)

Req (LBA: 0x07, Size: 1, DIR: W, Data: D)

Page Buffer

A	B	C	

LPA	PPA
0x00	–
0x01	0x01
…	–
0x04	0x00
…	–
0x07	–
…	…



Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

Req (LBA: 0x04, Size: 1, DIR: W, Data: A)

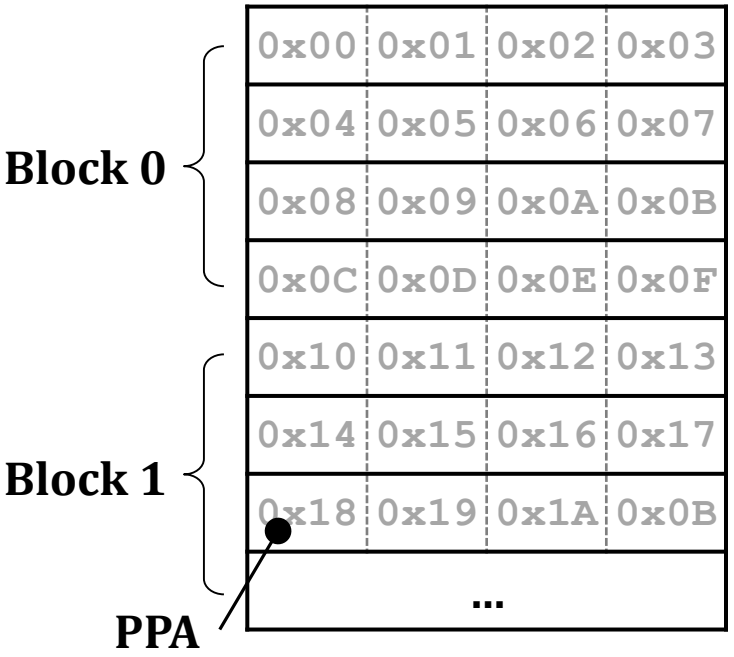
Req (LBA: 0x01, Size: 2, DIR: W, Data: B, C)

Req (LBA: 0x07, Size: 1, DIR: W, Data: D)

Page Buffer

A	B	C	

LPA	PPA
0x00	–
0x01	0x01
…	–
0x04	0x00
…	–
0x07	–
…	…



Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

Req (LBA: 0x04, Size: 1, DIR: W, Data: A)

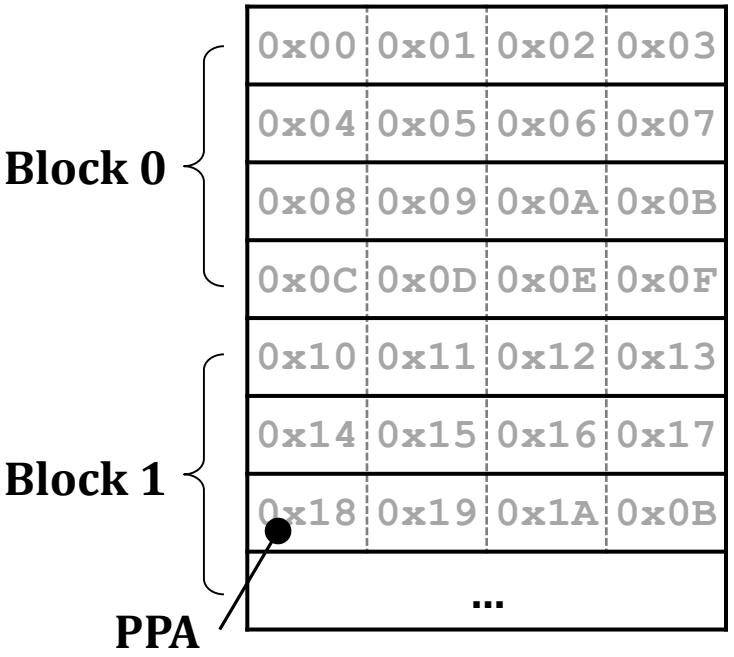
Req (LBA: 0x01, Size: 2, DIR: W, Data: B, C)

Req (LBA: 0x07, Size: 1, DIR: W, Data: D)

Page Buffer

A	B	C	D

LPA	PPA
0x00	–
0x01	0x01
…	–
0x04	0x00
…	–
0x07	–
…	…



Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

Req (LBA: 0x04, Size: 1, DIR: W, Data: A)

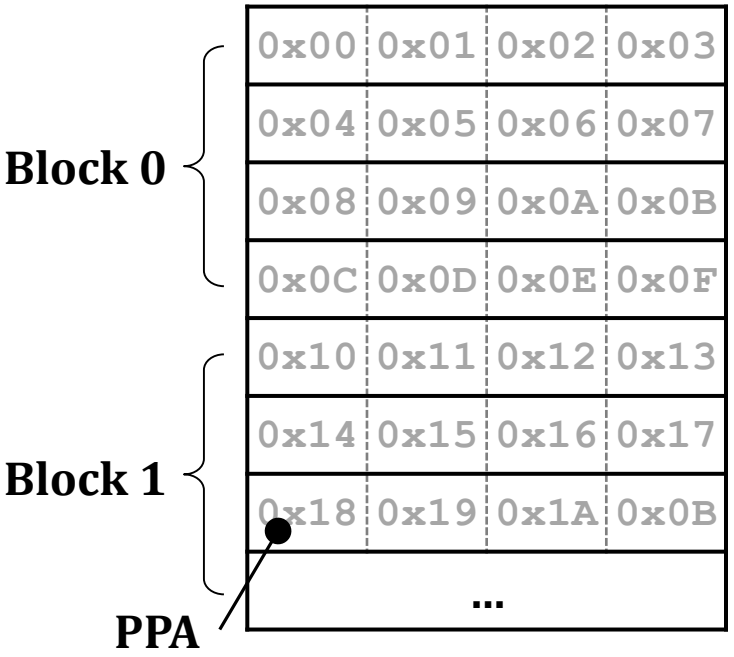
Req (LBA: 0x01, Size: 2, DIR: W, Data: B, C)

Req (LBA: 0x07, Size: 1, DIR: W, Data: D)

Page Buffer

A	B	C	D

LPA	PPA
0x00	–
0x01	0x01
…	–
0x04	0x00
…	–
0x07	0x03
…	…



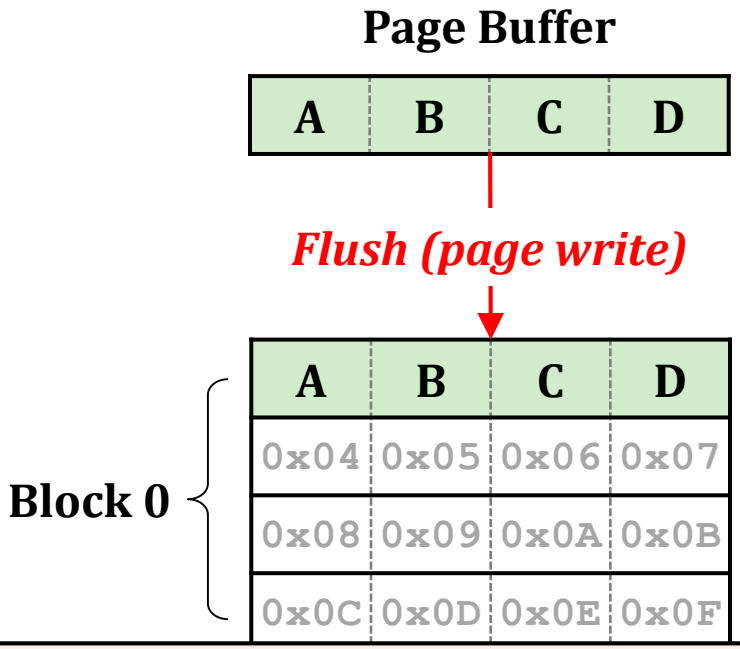
Fine-Grained Mapping + Page Buffer

- Write a page only when there are sufficient data blocks

Req (LBA: 0x04, Size: 1, DIR: W, Data: A)

Req (LBA: 0x01, Size: 2, DIR: W, Data: B, C)

Req (LBA: 0x07, Size: 1, DIR: W, Data: D)



LPA	PPA
0x00	–
0x01	0x01
…	–

Fine-grained mapping significantly reduces the number of NAND flash operations:

3 writes (+1 read) → 1 writes

Drawbacks of Fine-Grained Mapping

- Larger mapping table
  - 16-KiB mapping → 4 bytes per 16-KiB page = 0.025%
  - 4-KiB mapping → 4 bytes per 4-KiB page = 0.1%
  - For a 2-TB SSD, 2-GB DRAM is required.
    - Increases the SSD's price and power/energy consumption

- Data durability of written data
  - Page buffers are implemented by using volatile memory (e.g., SRAM or DRAM).

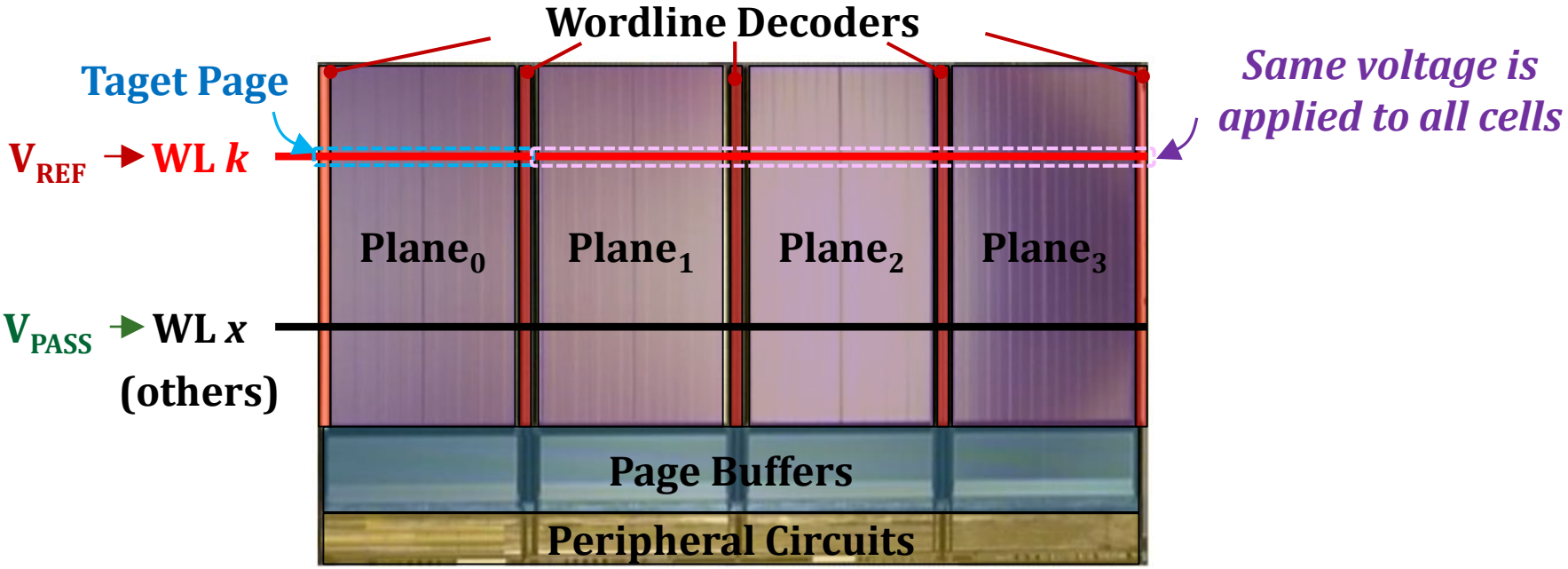
Despites non-negligible drawbacks, fine-grained mapping is widely used in modern SSDs due to its high benefits

Today's Agenda

- Fine-Grained Mapping
- Multi-plane Operation-Aware Blk. Mgmt.

Recap: Multi-Plane Operations

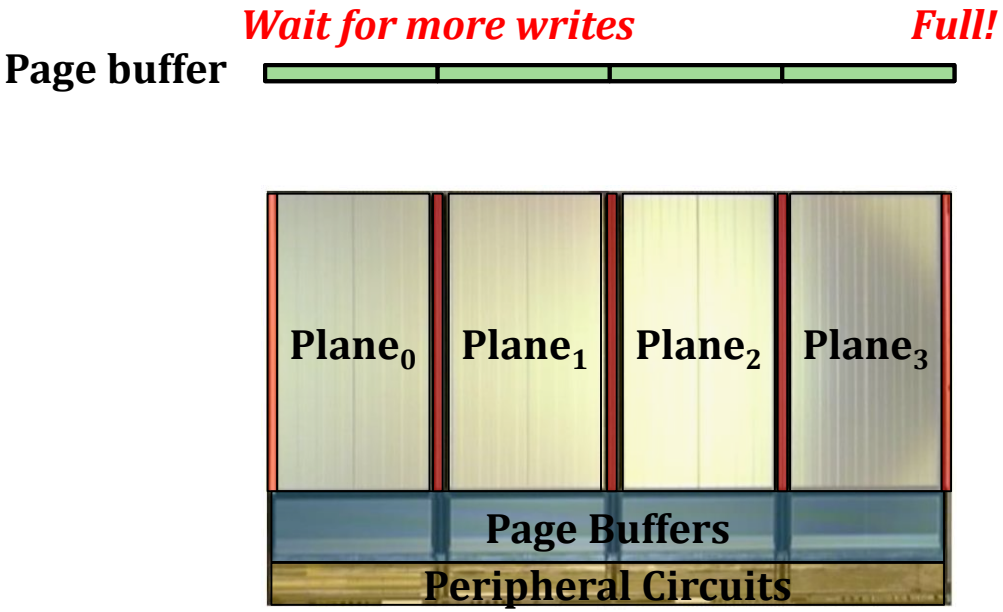
- Concurrent operations on different planes
  - Recall: Planes share WLs and row/column decoders



- Opportunity: Planes can concurrently operate
- Constraints: Only for the same operations on the same page offset

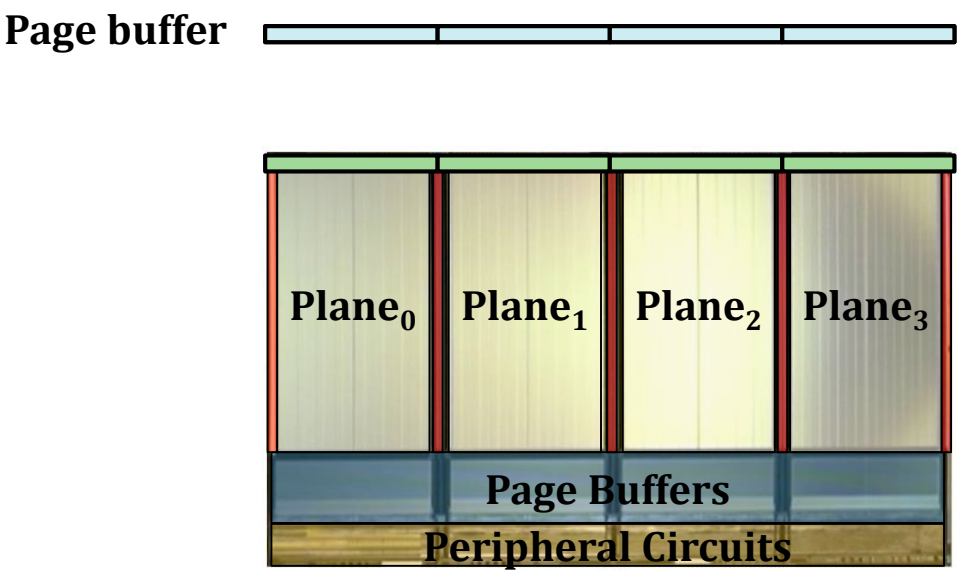
Multi-Plane-Aware Data Placement

- To perform as many multi-plane operations as possible
  - Flush N_{plane} pages at once after buffering them



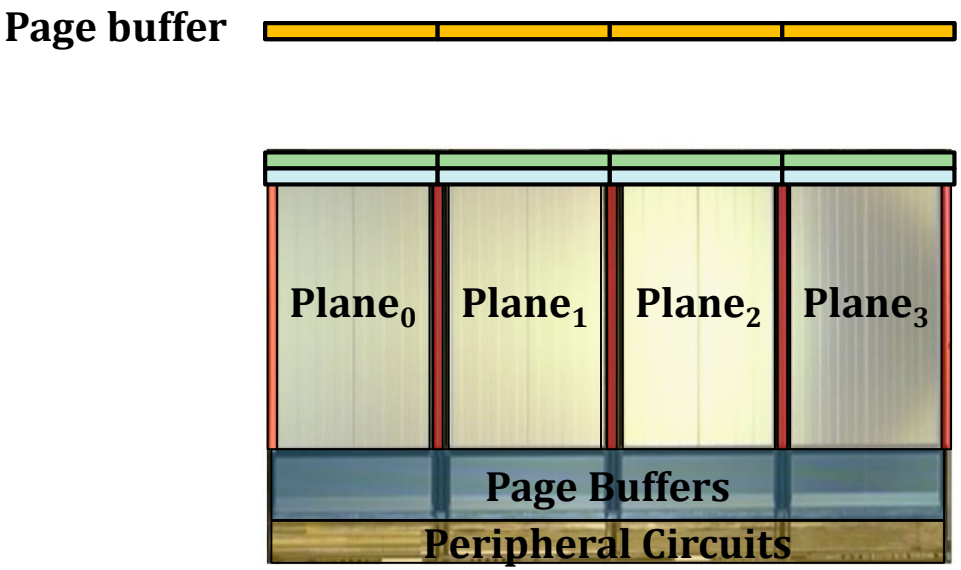
Multi-Plane-Aware Data Placement

- To perform as many multi-plane operations as possible
  - Flush N_{plane} pages at once after buffering them



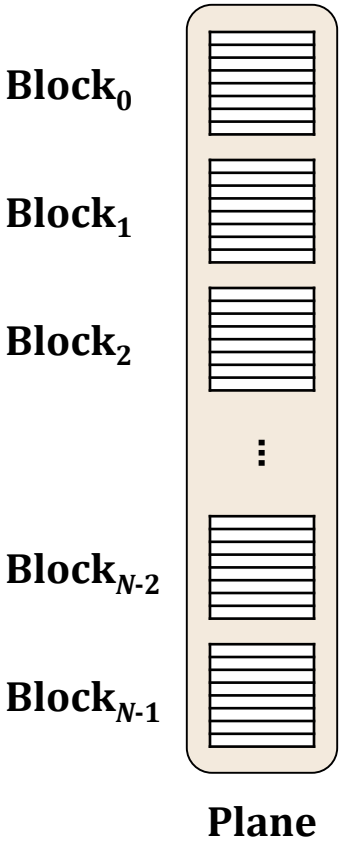
Multi-Plane-Aware Data Placement

- To perform as many multi-plane operations as possible
  - Flush N_{plane} pages at once after buffering them
  - Need to keep the write points of all planes to be the same
    - Superblock-based block management



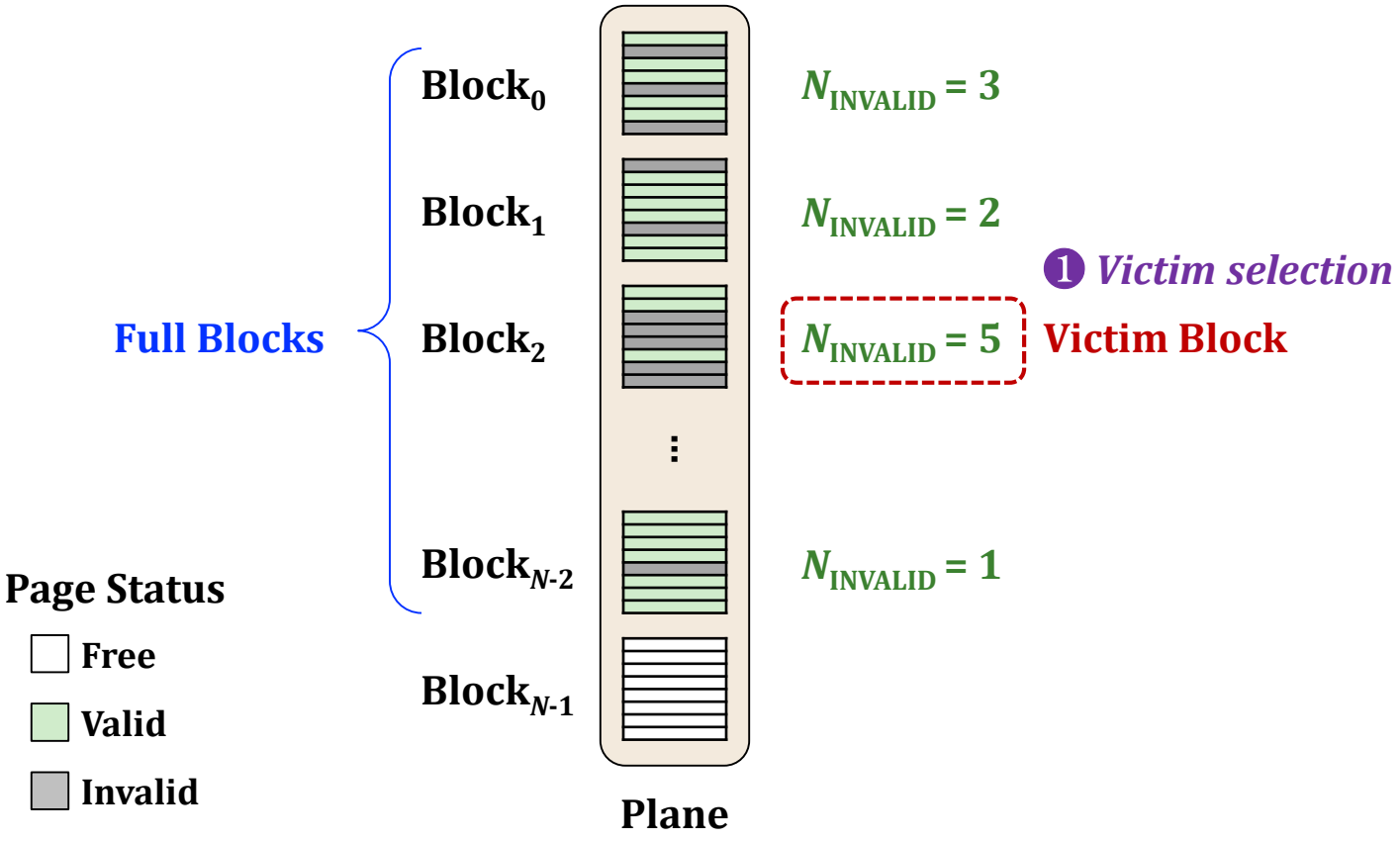
Multi-Plane-Aware Block Management

- Recap: For reducing the performance overhead of garbage collection, the FTL can select the block with the largest number of invalid pages (called a greedy policy).



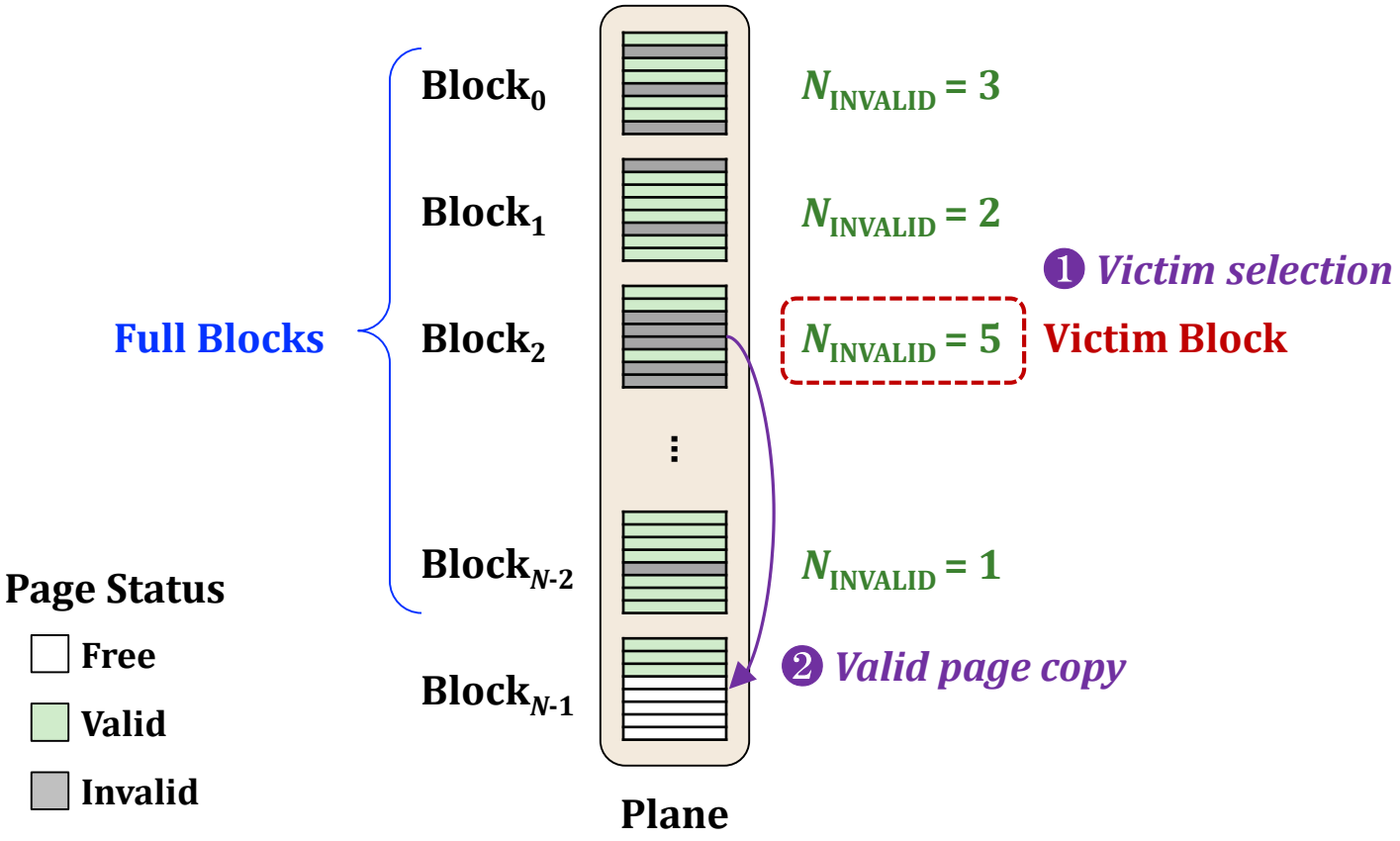
Multi-Plane-Aware Block Management

- Recap: For reducing the performance overhead of garbage collection, the FTL can select the block with the largest number of invalid pages (called a greedy policy).



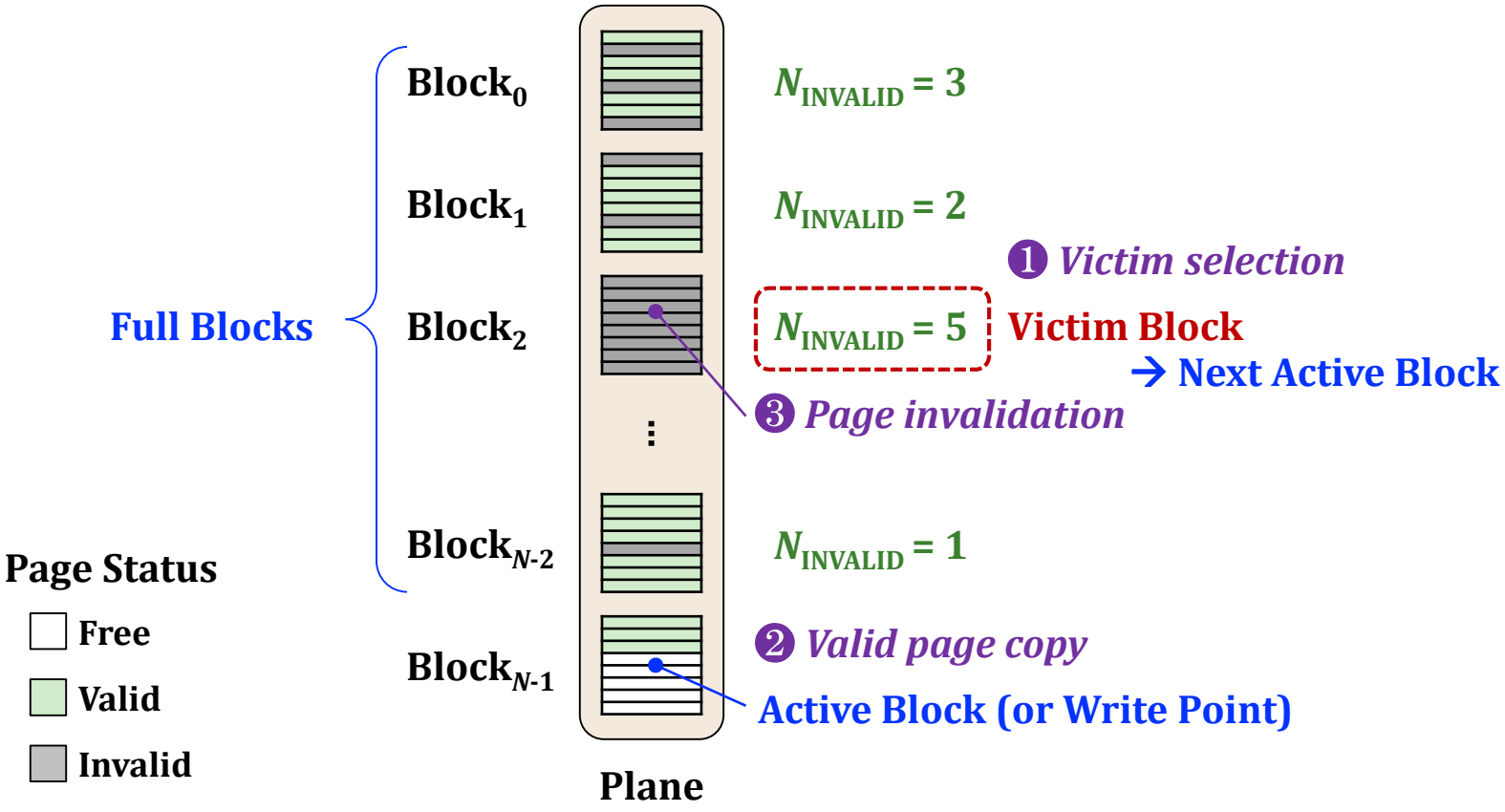
Multi-Plane-Aware Block Management

- Recap: For reducing the performance overhead of garbage collection, the FTL can select the block with the largest number of invalid pages (called a greedy policy).



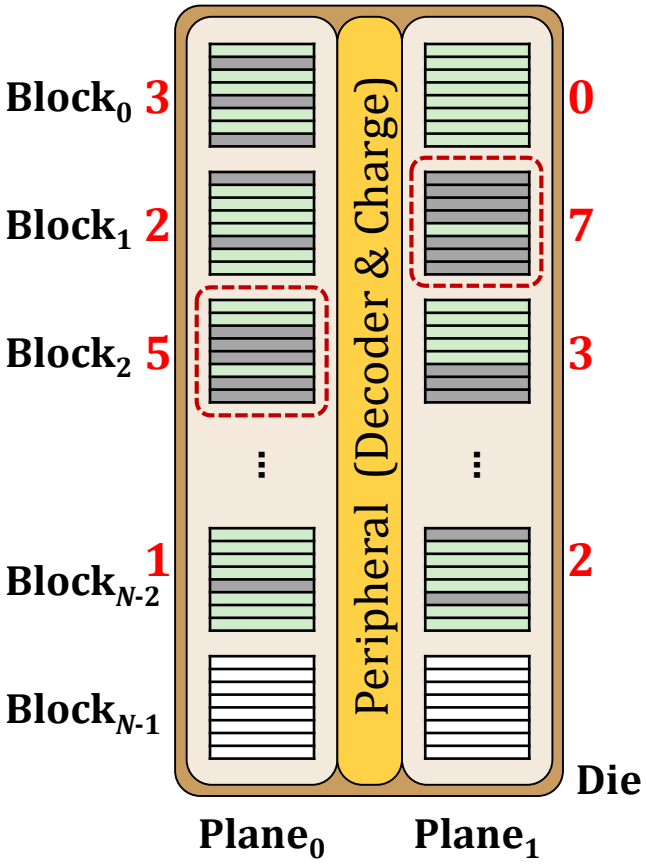
Multi-Plane-Aware Block Management

- Recap: For reducing the performance overhead of garbage collection, the FTL can select the block with the largest number of invalid pages (called a greedy policy).



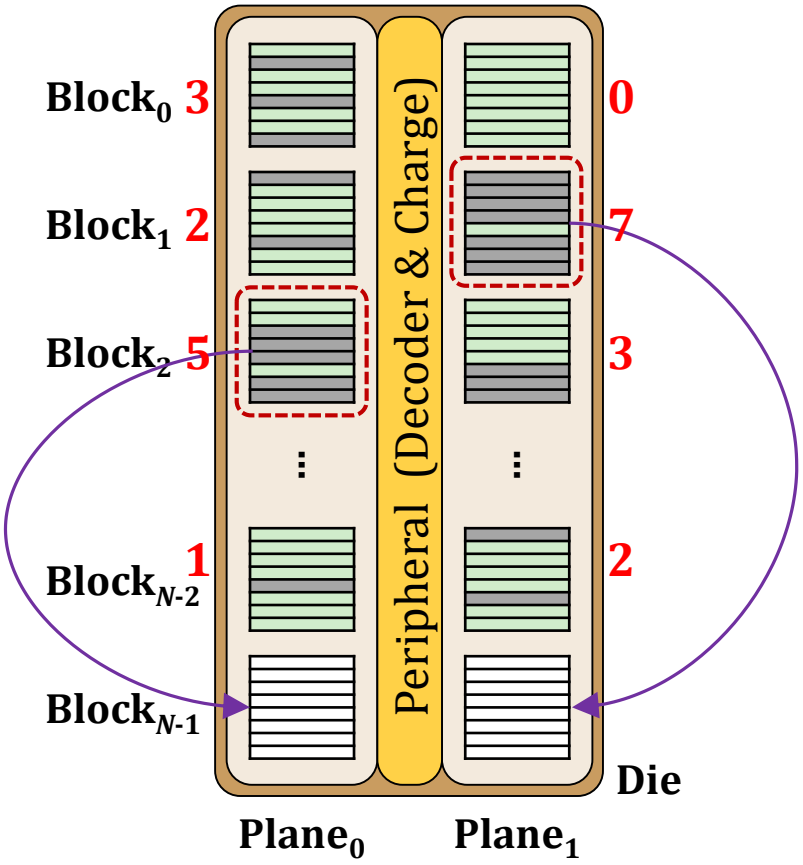
Multi-Plane-Aware Block Management

- Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.



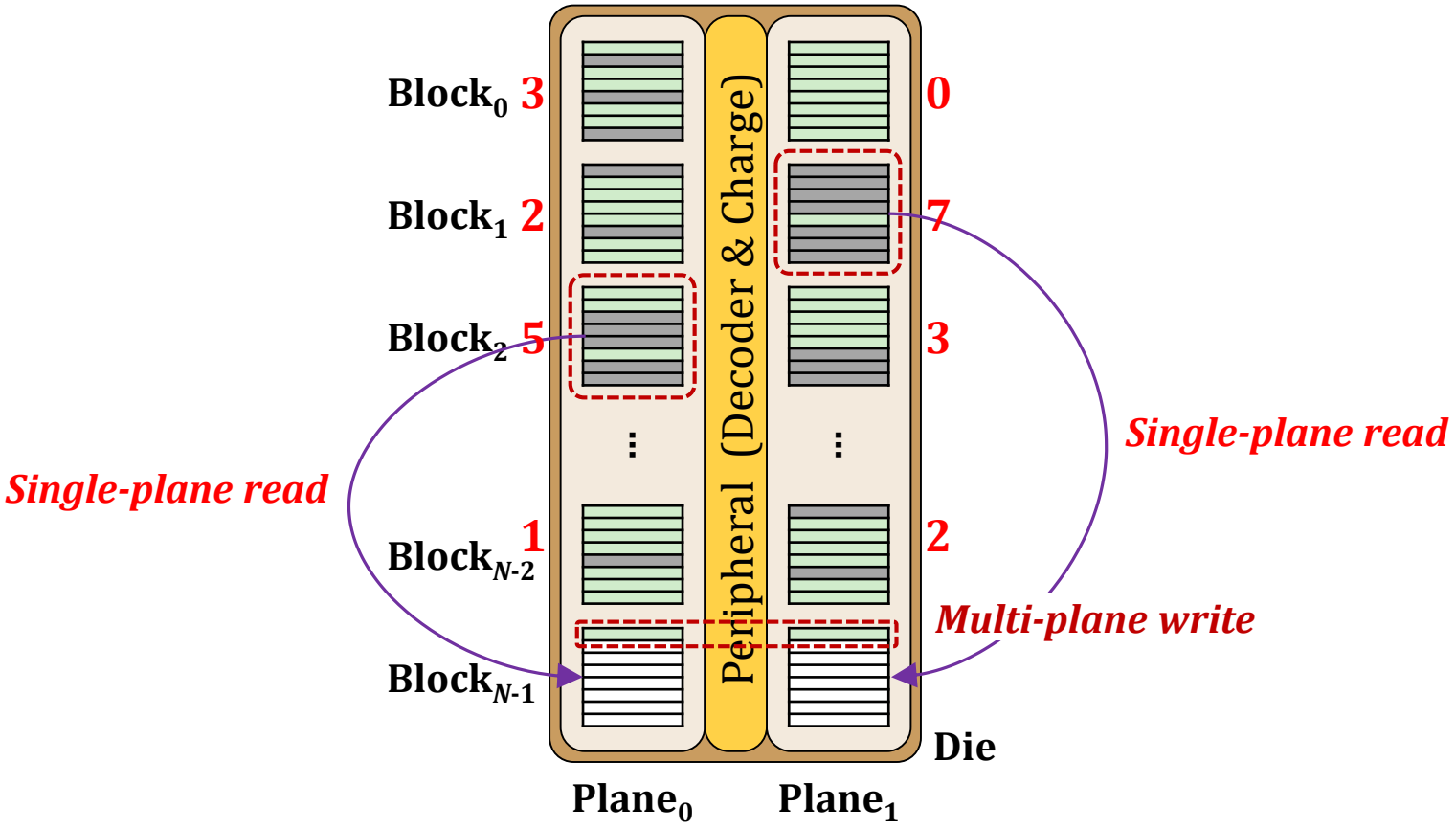
Multi-Plane-Aware Block Management

- Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.



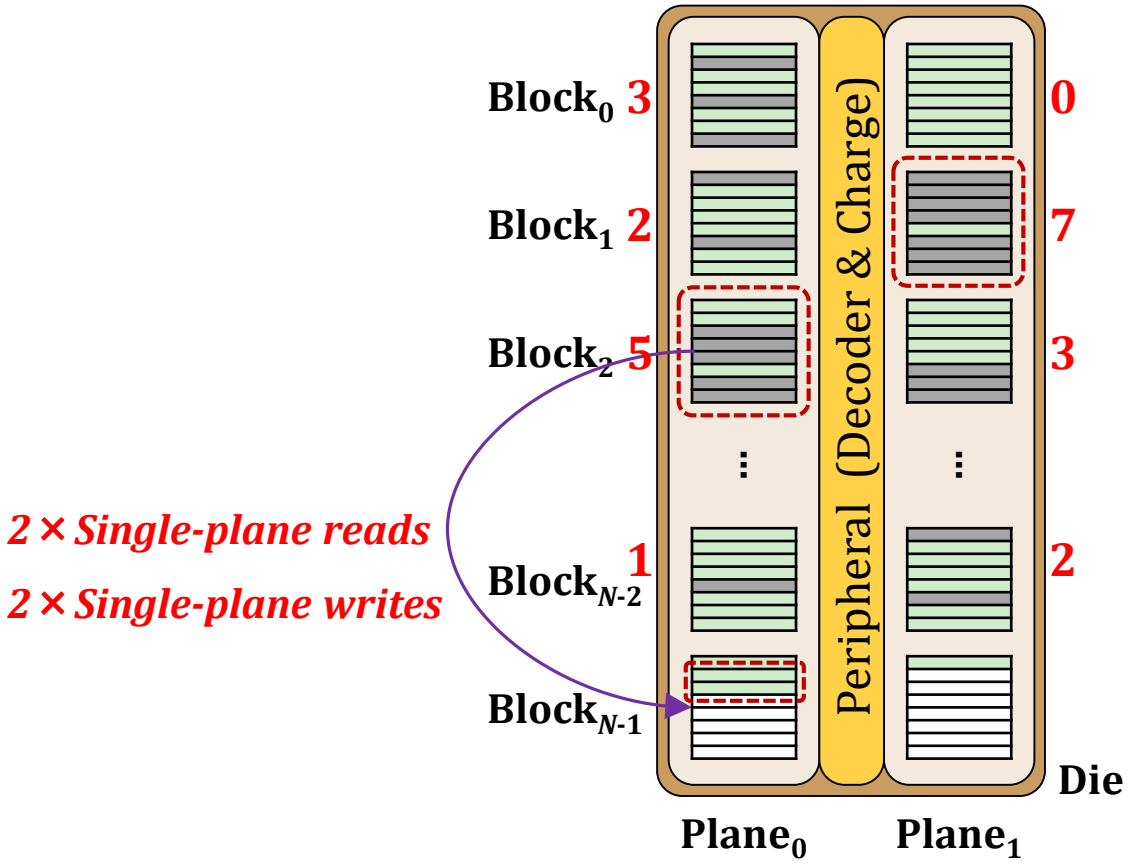
Multi-Plane-Aware Block Management

- Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.



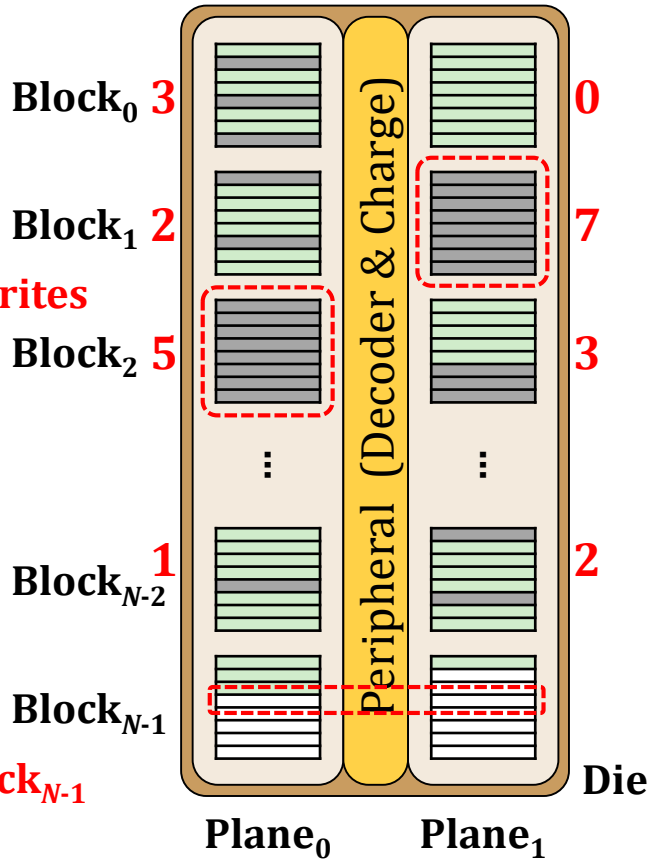
Multi-Plane-Aware Block Management

- Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.



Multi-Plane-Aware Block Management

- Recap: Planes in the same die can operate in parallel, but only when the page offsets are the same.



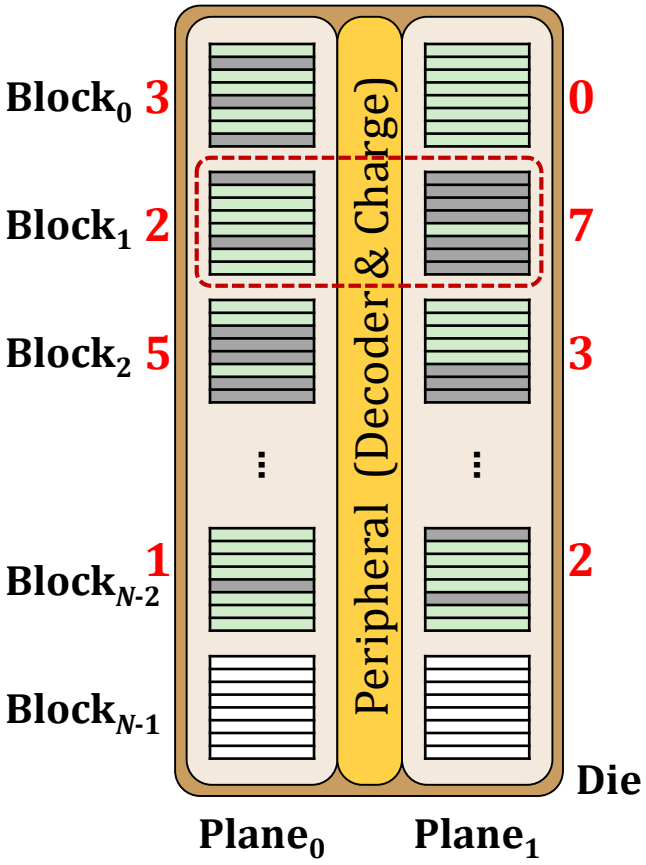
Cannot perform multi-plane writes for future writes (due to different offsets)

Cannot perform multi-plane writes before using (or discarding) 2nd and 3rd pages of Block$_{N-1}$ in Plane$_1$

4-page copy:
can be done with
2 × multi-plane reads &
2 × multi-plane writes
→ 4 × sigle-plane reads &
1 × multi-plane write &
2 × single-plane writes

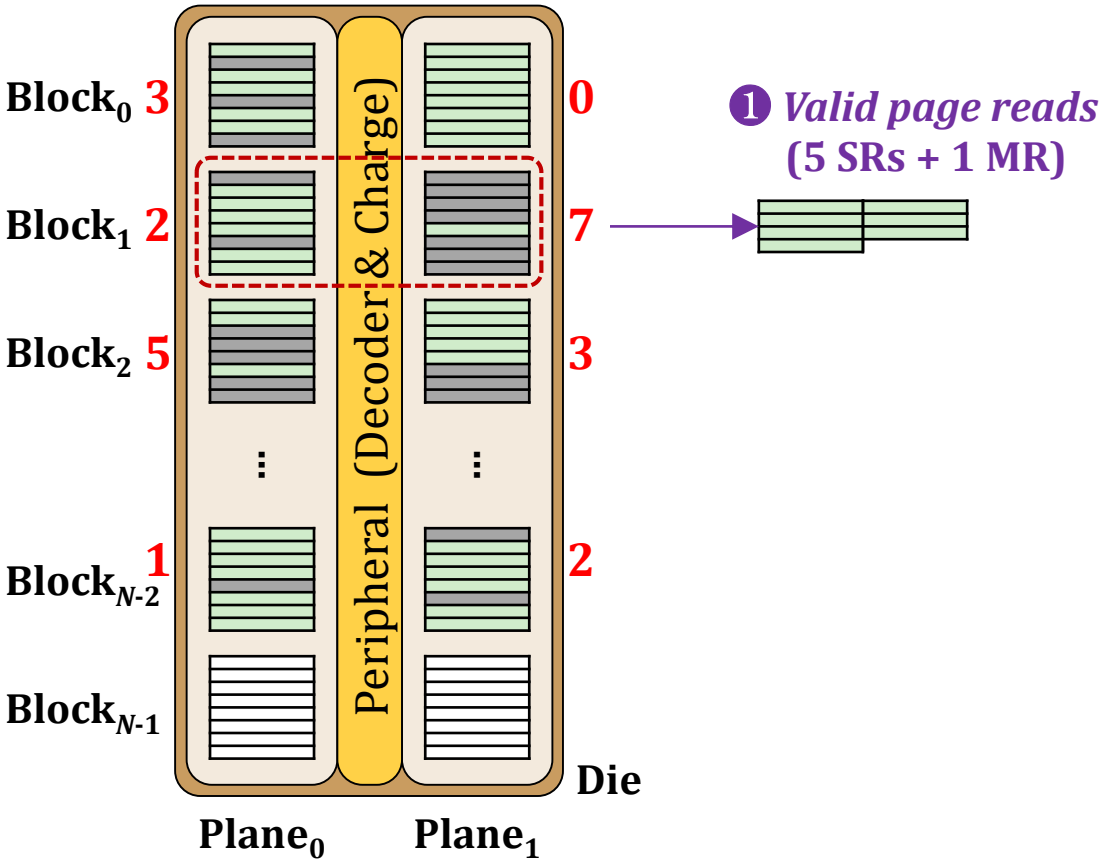
Multi-Plane-Aware Block Management

- Superblock-based management: groups each block with the same index (i.e., vertical position) in different planes



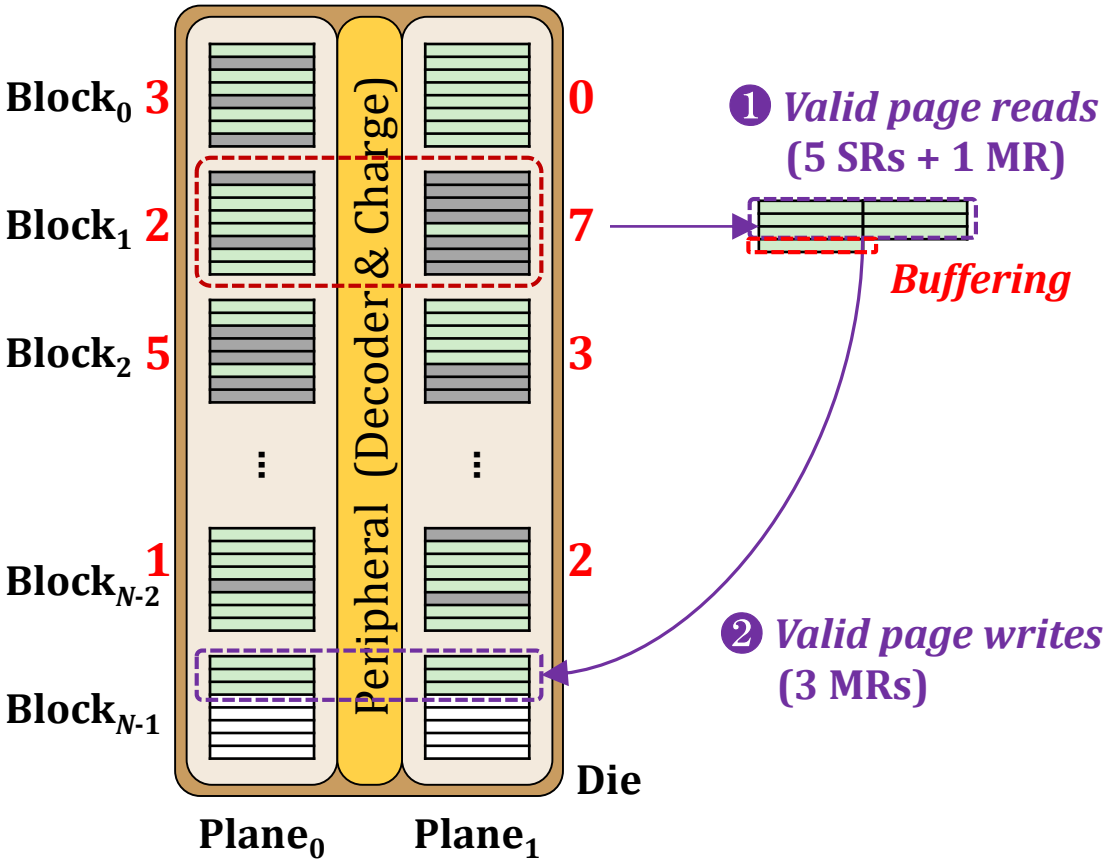
Multi-Plane-Aware Block Management

- Superblock-based management: groups each block with the same index (i.e., vertical position) in different planes



Multi-Plane-Aware Block Management

- Superblock-based management: groups each block with the same index (i.e., vertical position) in different planes

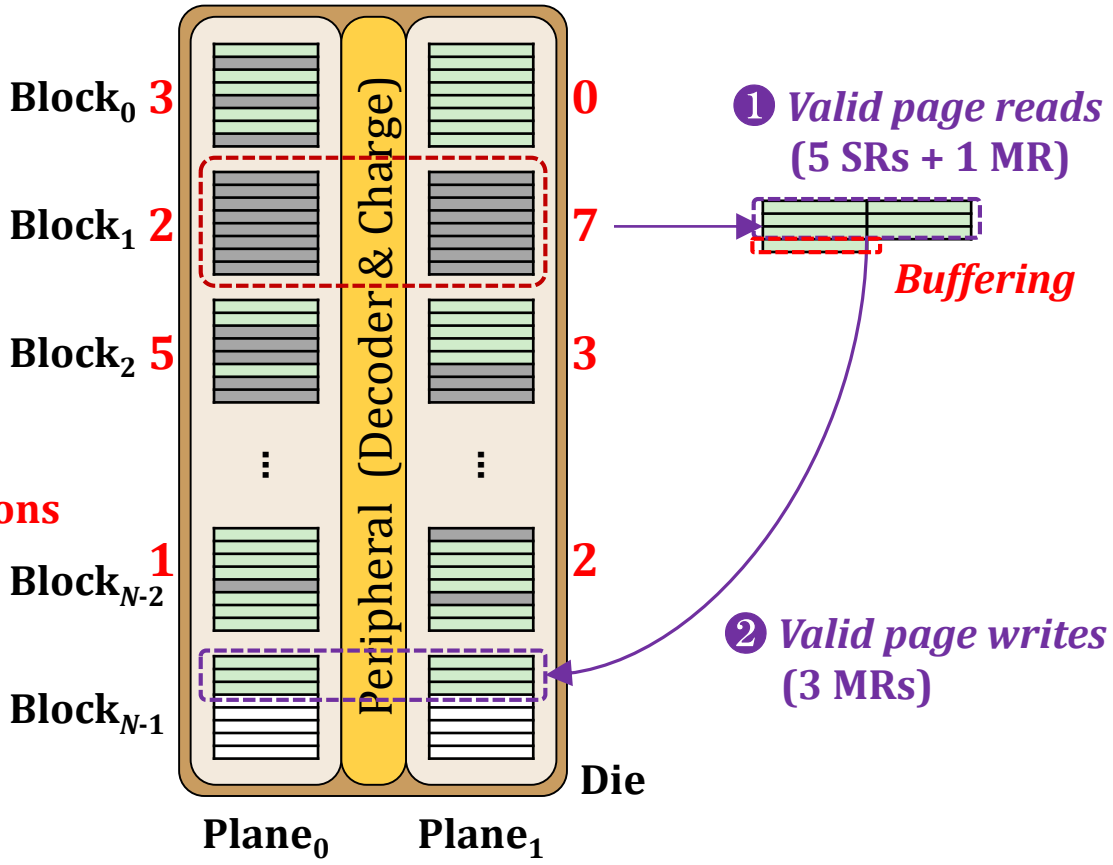


Multi-Plane-Aware Block Management

- Superblock-based management: groups each block with the same index (i.e., vertical position) in different planes

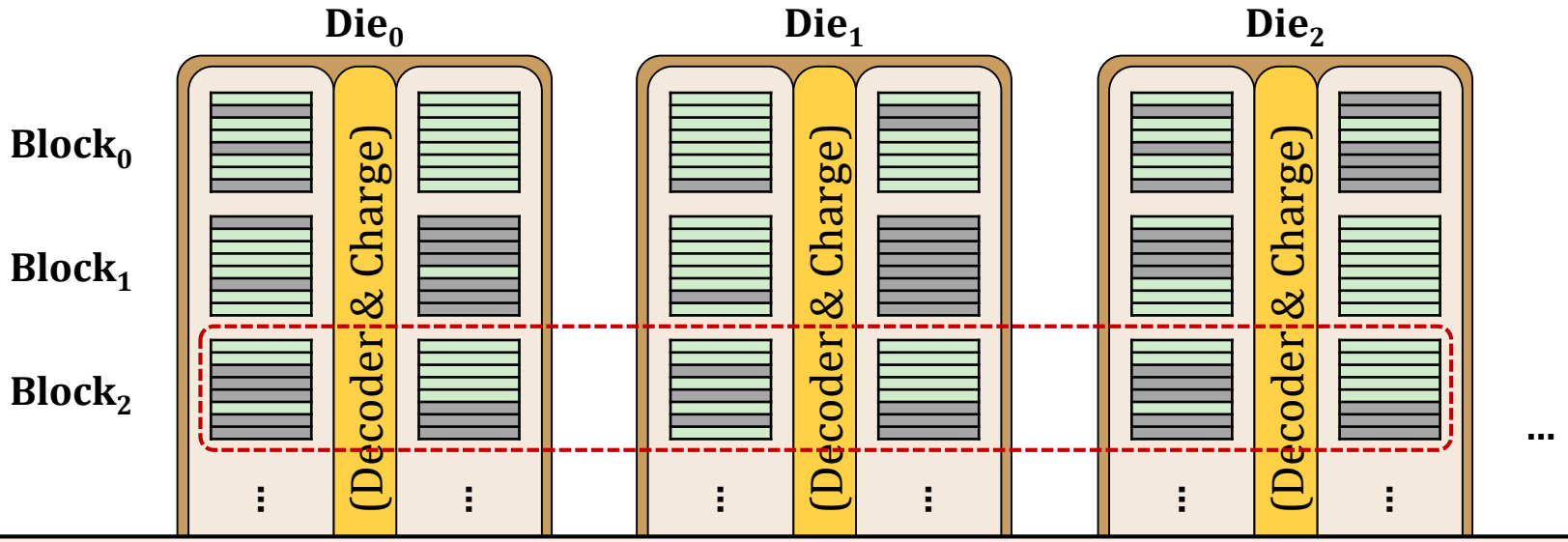
Pros:
Keep performing multi-plane writes

Cons:
More read/write operations
→ 5 SRs + 1 MR + 3 MWs
vs. 4 SRs + 1 MW + 2 SWs



Multi-Plane-Aware Block Management

- Offset management: Die level or SSD level?



Multi-plane operations can significantly improve SSD performance, but requires proper management in FTL

P&S Modern SSDs

Fine-Grained Mapping & Multi-Plane Operation-Aware Block Management

Dr. Mohammad Sadrosadati
Prof. Onur Mutlu
ETH Zürich

Fall 2022

14 December 2022